Math 116: Problem Set 8

Owen Jones

March 15, 2024

1. Suppose plaintext $P = L_0R_0$ encrypts to ciphertext C. Recall the structure of a Feistel cipher:

   $$L_i = R_{i-1} \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

   Consider plaintext $\overline{P} = \overline{L_0R_0}$. It suffices to show that after each round i, the output resulting from initial plaintext $\overline{P}$ and key $\overline{K_i}$ is the complement of the output resulting from initial plaintext P and key K_i.

   $$\overline{R_0} \oplus \overline{K_1} = R_0 \oplus 11111\ldots \oplus K_1 \oplus 11111\ldots = R_0 \oplus K_1$$
   $$\Rightarrow f(\overline{R_0}, \overline{K_1}) = S(\overline{R_0} \oplus \overline{K_1}) = S(R_0 \oplus K_1) = f(R_0, K_1) \text{ (inputs are the same)}$$
   $$L_1' = \overline{R_0} = \overline{L_1} \quad R_1' = \overline{L_0} \oplus f(\overline{R_0}, \overline{K_1}) = \overline{L_0} \oplus f(R_0, K_1) = \overline{R_1}$$
   $$\Rightarrow L_1'R_1' = \overline{L_1R_1}$$

   Suppose after i rounds $L_i'R_i' = \overline{L_iR_i}$.

   $$L_{i+1}' = \overline{R_i} = \overline{L_{i+1}} = \quad R_{i+1}' = \overline{L_i} \oplus f(\overline{R_i}, \overline{K_{i+1}}) = \overline{L_i} \oplus f(R_i, K_{i+1}) = \overline{R_{i+1}}$$
   $$\Rightarrow L_{i+1}'R_{i+1}' = \overline{L_{i+1}R_{i+1}}$$

   Thus, by induction, the output resulting from initial plaintext $\overline{P}$ and key $\overline{K_i}$ is the complement of the output resulting from initial plaintext P and key K_i for all i. Hence, $\overline{P}$ encrypts to ciphertext $\overline{C}$.

2. (a) Suppose $x_1 \oplus x_2 = x_3 \oplus x_4$. Because XOR is linear

   $$f(x_1) \oplus f(x_2) = \alpha x_1 + \beta \oplus \alpha x_2 + \beta = \alpha(x_1 \oplus x_2) + \beta \oplus \beta$$
   $$= \alpha(x_3 \oplus x_4) + \beta \oplus \beta = \alpha x_3 + \beta \oplus \alpha x_4 + \beta = f(x_3) \oplus f(x_4)$$

   , so f has the equal difference property.

   (b) Suppose $x_1 \oplus x_2 = x_3 \oplus x_4$.

   Shiftrow Consider row 2 of the ShiftRow matrix for input x_k $c_2^{(k)} = \begin{bmatrix} c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \end{bmatrix} = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \end{bmatrix}$. Observe $c_2^{(1)} \oplus c_2^{(2)} = c_2^{(3)} \oplus c_2^{(4)}$ because $b_{i,j}^{(1)} \oplus b_{i,j}^{(2)} = b_{i,j}^{(3)} \oplus b_{i,j}^{(4)}$ will hold for each byte in the 4×4 matrix. Similarly, this will hold for any other row, so the output bit-strings $c^{(1)} \oplus c^{(2)} = c^{(3)} \oplus c^{(4)}$ will have this property.

Mixcolumn In part (a) we showed that an affine function has the equal difference property. For any byte in the output matrix, we have $d_{i,j}^{(k)} = \alpha_{i,j} c_{i,j}^{(k)}$ for some $alpha_{i,j} \in \mathbb{F}_{2^8}$. Thus, $d_{i,j}^{(1)} \oplus d_{i,j}^{(2)} = d_{i,j}^{(3)} \oplus d_{i,j}^{(4)}$ for every byte, so the output bit-strings $d^{(1)} \oplus d^{(2)} = d^{(3)} \oplus d^{(4)}$ will have this property.

RoundKey $k_{i,j} \oplus d_{i,j}^{(1)} \oplus k_{i,j} \oplus d_{i,j}^{(2)} = d_{i,j}^{(1)} \oplus d_{i,j}^{(2)} = d_{i,j}^{(3)} \oplus d_{i,j}^{(4)} = k_{i,j} \oplus d_{i,j}^{(3)} \oplus k_{i,j} \oplus d_{i,j}^{(4)}$ for each byte in the 4×4 matrix because $k_{i,j} \oplus k_{i,j} = 00000\ldots$ and XOR is commutative. It follows this must also hold for the output bit-string.

3. (a) In $2b$ we showed that each of ShiftRow, MixColumn, and RoundKey have the equal difference property, so it's pretty trivial that their composition would have the equal difference property. $x_1 \oplus x_2 = x_3 \oplus x_4 \Rightarrow f(x_1) \oplus f(x_2) = f(x_3) \oplus f(x_4) \Rightarrow g(f(x_1)) \oplus g(f(x_2)) = g(f(x_3)) \oplus g(f(x_4))$ if both f and g have the equal difference property.

   (b) In $2b$ we showed $k_{i,j} \oplus d_{i,j}^{(1)} \oplus k_{i,j} \oplus d_{i,j}^{(2)} = d_{i,j}^{(1)} \oplus d_{i,j}^{(2)}$ for each byte in the 4×4 matrix. Thus, $E(x_1) \oplus E(x_2)$ is only dependent on the ShiftRow and MixColumn steps.

   (c) We can exploit the fact that $E(x_1) \oplus E(x_2)$ is independent of the key. We can apply InvMixColumn and InvShiftRow to $E(x_1) \oplus E(x_2)$ to obtain $x_1 \oplus x_2$. This is also guaranteed by the equal difference property. We know x_1, so $x_1 \oplus x_1 \oplus x_2 = x_2$.

4. We have $x_1 \oplus x_2 = x_3 \oplus x_4$. By $2a$, we would have $BS(x_1) \oplus BS(x_2) = BS(x_3) \oplus BS(x_4)$ if the ByteSub transformation was an affine map. However, we have a counterexample where that is not the case, so ByteSub transformation cannot be an affine map.

5. $P_{j+1} = D_K(C_{j+1}) \oplus C_j$ and $P_j = D_K(C_j) \oplus C_{j-1}$ will be decrypted incorrectly because they are all of the blocks that depend on C_j.

6. Suppose x and y are two messages such that $x \equiv y \pmod{p-1}$. Then $h(x) = h(y)$, so we will have multiple messages with the same hash value. Moreover, if it's not computationally difficult to find 2 messages with the same hash value.

7. (a) $a_0 = 1$

   (b) $x \equiv 5 \pmod{8}$

   (c) $b_0 = 1$ $b_1 = 2$ $x \equiv 7 \pmod{9}$

   (d) $x \equiv 7 \pmod{13}$

   (e) $x = 709$ by CRT

8. $x \equiv 2 \pmod{4}$, $x \equiv 14 \pmod{27}$ $x \equiv 7 \pmod{11}$ By CRT $x \equiv 986 \pmod{1188}$

In [1]:
```
import math116
import numpy as np
```

In [2]:
```
p=937
```

In [3]:
```
n=p-1
```

In [4]:
```
math116.factor(n)
```

Out[4]:
```
Counter({2: 3, 3: 2, 13: 1})
```

In [5]:
```
y=46
```

In [8]:
```
pow(y,n//2,p)
```

Out[8]:
```
936
```

In [39]:
```
y_1=(y*pow(5,-1,p))%p
```

In [12]:
```
y_1
```

Out[12]:
```
384
```

In [14]:
```
pow(y_1,n//8,p)
```

Out[14]:
```
936
```

In [45]:
```
h=pow(5,n//13,p)
h
```

Out[45]:
```
911
```

In [46]:
```
[pow(h,i,p) for i in range(13)]
```

Out[46]:
```
[1, 911, 676, 227, 657, 721, 931, 156, 629, 512, 743, 359, 36]
```

In [40]:
```
pow(y_1,n//9,p)
```

Out[40]:
```
322
```

In [47]:
```
pow(y,n//13,p)
```

Out[47]:
```
156
```

In [49]:
```
math116.crt_general([5,7,7],[8,9,13])
```

Out[49]:
```
(709, 936)
```

In [50]:
```
pow(5,709,p)
```

Out[50]:
```
46
```

In [99]:
```
y=14652320651439828423046368044446485954319801736
```

In [91]:
```
p=1047073721667575963973626541914699579101292247109
```

In [54]:
```
n=p-1
```

In [92]:
```
h=pow(13,n//3,p)
h
```

Out[92]:
```
1039811625092996309995927348674998798108568888626
```

In [93]:
```
[pow(h,i,p) for i in range(3)]
```

Out[93]:
```
[1,
 1039811625092996309995927348674998798108568888626,
 7262096574579653977699193239700780992723358482]
```

In [89]:
```
pow(y,n//3,p)
```

Out[89]:
```
7262096574579653977699193239700780992723358482
```

In [94]:
```
y_1=(y*pow(13,-2,p))%p
y_1
```

Out[94]:
```
365633147331588294099804806751548648833908653143
```

In [95]:
```
pow(y_1,n//9,p)
```

Out[95]:
```
1039811625092996309995927348674998798108568888626
```

In [96]:
```
y_2=(y*pow(13,-5,p))%p
```

In [97]:
```
pow(y_2,n//27,p)
```

Out[97]:
```
1039811625092996309995927348674998798108568888626
```

In [87]:
```
math116.crt_general([2,14,7],[4,27,11])
```

Out[87]:
```
(986, 1188)
```

In [98]:
```
q=881375186588868656543456685113383484091996841
```

In [101…
```
 pow(y,n//2,p)
```

Out[101…
```
1
```

In [102…
```
pow(13,n//2,p)
```

Out[102…
```
1047073721667575963973626541914699579101292247108
```

In [ ]: